2013

Data warehouse modeler (A CASE Tool): redesign and implementation of framework

Geetha Tummala

Data Warehouse Modeler (A CASE Tool): Redesign and Implementation of Framework

by

Geetha Tummala

A creative component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Interdisciplinary Graduate Studies

Program of Study Committee:

Dr. Sree Nilakanta, Major Professor

Dr. Leslie Miller Co-major Professor

Dr. Manimaran Govindarasu

Iowa State University
Ames, Iowa
2013

Table of Contents

Abstract

Data warehousing is an essential element of decision support and is widely used today in many large organizations that deal with a massive amount of information. It hastens the process of retrieving information needed for decision making. In order to supply a decisional database, a meta-data is needed to enable the communication between various functional areas of the warehouse and an ETL tool (Extraction, Transformation, and Load) to define the warehousing process. Thus, a CASE tool named "Data Warehouse Query Translation Interface" (DWQTI) has been designed to generate SQL queries necessary to build a warehouse from a set of operational relational databases. The warehouse designer simply specifies a list of attribute names that will appear in the warehouse conditions, if any are desired; and a description of the operational databases. The tool then returns the SQL query needed to populate the warehouse table [1]. This creative component project is to refactor and implement an improvised framework of existing Data Warehouse Query Translation Interface (DWQTI) with additional functionalities; a legacy application implemented in Java to C# under Net framework.

Keywords: Data warehouse, metadata, SQL queries, mapping expression

1. Introduction

Global corporations today compete with each other in satisfying customers through introducing better services and performance in the most cost effective way. Millions of transactions had been captured through the years in enterprises producing enormous amounts of raw data spread among different distant departments [6, 7]. Earlier, operational database systems also, known as On-Line Transactional Processing (OLTP) systems were used for decision making and report generation. As the need for better and faster decision support emerged, the OLTP systems proved to be deficient. This drawback urged database experts to construct

databases in a way to support analysis and decision making and Data Warehouses evolved [6]. A data warehouse is segregated from transactional databases and contains consistent cleansed data. In order to satisfy rapid information retrieval and ad hoc formulations, modeling became an important step in a data warehouse design process, including logical data modeling, physical data modeling, and metadata management [1]. In addition to the logical data model, the warehouse design maintained a meta-database consisting of information that will aid in administering the warehouse.

Metadata is an enabling technology that supports the user interface to warehouse such as the ROLAP front-end tool, which makes use of metadata to display data warehouse tables and fields. Access tools that utilize metadata are a powerful evolution of a warehousing process. Miller and Nilakanta [2] introduced a software tool (DWQTI) to facilitate the design and development of warehouses, where the warehouse designer simply specifies a list of attribute names that will appear in the warehouse, conditions if any desired, and a description of the operational databases. Then, the tool returns queries needed to populate the warehouse table. The software tool can be viewed and used as a small component of a data warehouse.

DWQTI enables users obtain the desired operational database query information in SQL. But, user friendliness had been a significant design issue in this version of the tool. The UI design has been a low priority item in the existing java application with more focus given on the implementation and correctness of warehouse modeler. Also, with the emerging version of new software, extending or modifying the old legacy application is infeasible that triggers the need for transformation into newer software. The goal of this creative component is to address the UI design issues and to upgrade the existing legacy system to newer platform; while preserving the external behavior of the code yet improving its internal structure. Re-implementing application

on new platform can reduce operational costs, and the additional capabilities of new technologies can provide access to valuable functions such as Web Services and Integrated Development Environments. Once transformation is complete the application can be aligned more closely to current and future business needs through the addition of new functionality to the transformed application.

1.1 Background

Data Warehouse Query Translation Interface (DWQTI) program provides a user interface and a modeler system that allows the users to generate the desired operational database query information in SQL; after the user selects the warehouse attributes and compose the warehouse conditions on the interface.

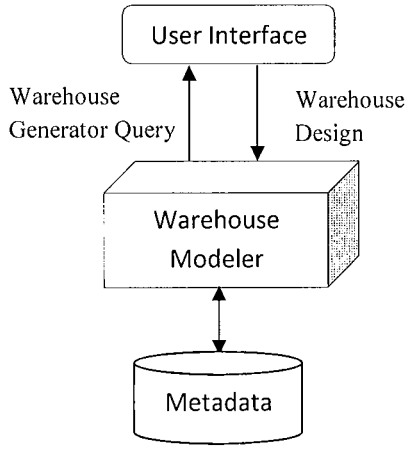

Figure 1: Three Tier Architecture for Data Warehouse (L., Miller, L., & Nilakanta, S. [2])

The DWQTI tool can be viewed and used as a small component of a data warehouse. Users do not have to know as which attribute resides in which relation or DB scheme. The underlying physical databases can be in remote sites; only the operational metadata files are stored locally so that the relation scheme names, attribute names, and possibly some functional dependencies for the underlying databases can be retrieved. The design of the modeler or the mapping engine of this

interface is based upon results in [3] [4] [5]. Owrang and Miller [3] presented the basic idea and the algorithms for processing query translations with the hypergraph model in a universal relational database system environment. Miller *et al.* [4] discussed the methods for testing the lossless join property of subhypergraphs. Lin *et al.* [5] introduced the design and the implementation for a Universal Relation Language Interface for INGRES using the algorithms formed in [3] and [4].

This creative component is introduced to optimize and refactor the framework of DWQTI tool. DWQTI is a standalone application that's designed and implemented in JAVA (JDK 1.0.2). It is implemented as a public class which extends the JAVA AWT class frame and uses several classes and methods; a total of fourteen user defined methods. Methods inherited from the AWT class frame, seven utility classes and six data structure classes are designed and implemented to support the fourteen methods. The implemented java program is a heavily loaded application.

In the next section, the introduction to the data warehouse model is discussed. Section 3 examines the modeler design and implementation of the existing DWQTI tool. The conceptual framework of the new tool is given in Section 4 and Section 5 looks at the future wok.

2. The Warehouse Model

Data warehouse is a read-intensive database, modeled according to enterprise requirements within different environments. It can be seen as building blocks involving a number of essential components glued together consisting of source data, data staging, data storage, and information delivery [7]. The metadata is used to combine and manage these blocks. Different data warehouse architectures had been conducted. Among them the most popular is Kimball's methodology which is also known as Bottom-Up approach.

The data warehousing technologies can be roughly classified into three categories: data access tools, database connectivity, and database technology. Using data access tools, the end-users can efficiently access the enterprise data. The success of a data warehouse largely depends upon whether data access tools can provide a high level of end-user productivity. On the other hand, database connectivity technology is crucial for the data warehouse design and development since the data warehouse designers and developers have to consider the issues such as integrating highly heterogeneous data sources. Finally, the need for high level and high quality data management in the data-warehousing environment challenges the database technology due to issues such as data replication management, query translation, and query optimization.

In the next section we look at the interface of the existing warehouse modeler that's developed in Java. The following section will describe the interface design and implementation of the new refactored CASE tool. The foundations and a current implementation of the warehouse modeler are examined in section 4 and the paper is concluded by looking at the future work associated with the new interface.

3. DWQTI Interface Design and Implementation – Java Application

3.1. Existing solution

In this section, an introduction to the design and the implementation of the user interface of the data warehouse modeler system that's developed using Java is given. The system is designed to work with a very simple view of the source data. The interface shows a list of attribute names to represent the data available in the data sources. In the event that names conflict, either views renaming the attributes is used or the names can be qualified. The metadata (e.g. functional dependencies, tables, etc.,) is assumed to have been supplied by the user or developed through simple extraction programs. The types and format of the metadata is not important in the context

of the presentation [1]. In Section 3.2, a general description, including a layout of the DWQTI interface is presented. In Section 3.3, the implementation of the user interface by looking at the functionality and relations of the class - DWQTI and relevant utility and data structure classes is discussed. Section 3.4 covers some examples of the tool.

3.2. Interface design

The current prototype of the DWQTI CASE tool, an UI interface for the data warehouse modeler has been implemented on HP-UX machine using Java language and tested across a few platforms such as UNIX (DEC Alpha workstations and HP-UX workstations), Window 95, Window NT 4.0, and Macintosh. The user interface of this system has been a low priority item with more focus given on the implementation and correctness of warehouse modeler. As a result, the system uses a command line format for the user interface. User friendliness had been a substantial design issue in the next version of this tool. Accordingly, recent studies show that user interface effectiveness has become increasingly important in software development, as the user interface has the power to "make or break" a software product [9]; hence re-designing of UI has become a primary factor during this application migration.

The layout as shown in Fig. 2, has six functional regions. In the following subsections, the functionality of each UI region is explained.

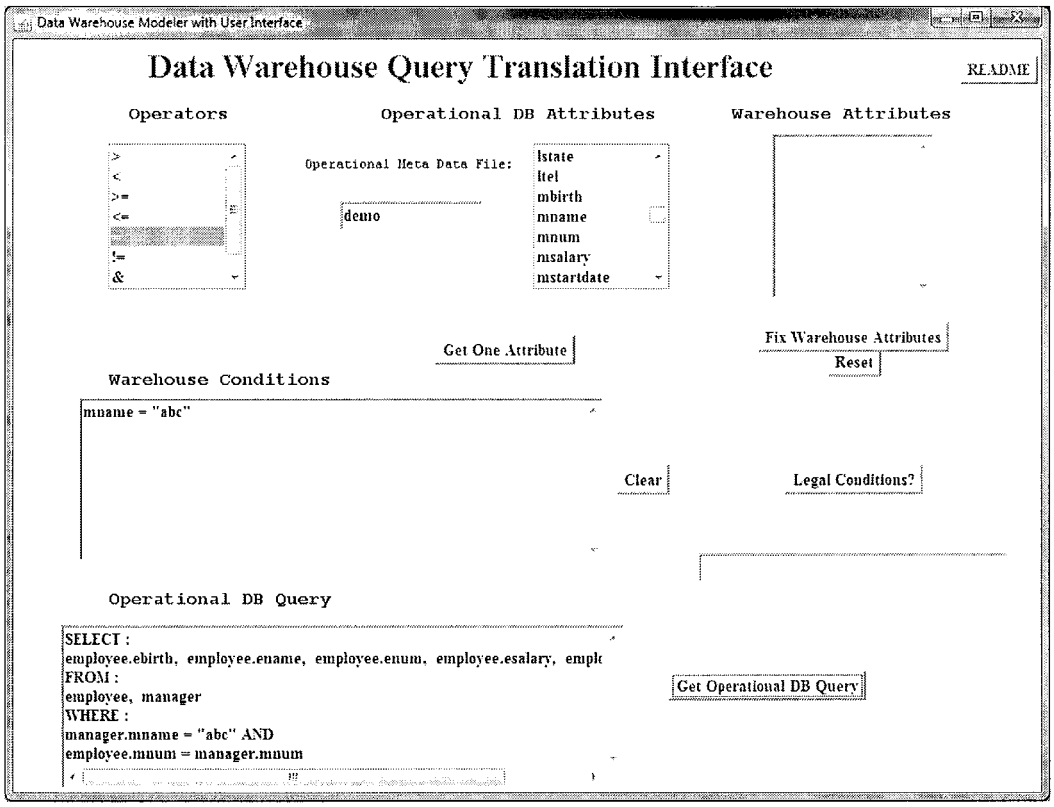

Figure 2: Data Warehouse Query Translation Interface – Java System (Wu, L., Miller, L., & Nilakanta, S)

3.2.1. Operational DB attributes

The operational DB attributes region of the UI handles the metadata file. Before using the case tool, the user must ensure that the design information on the operational relational database(s) has been stored in the case tool's file system. When the information on the operational relational database(s) has been stored, the user can start the warehouse query generation tool.

After typing in the operational metadata filename in the small text field labeled with "Operational Metadata File", a scrolling list of all attribute names in the given operational metadata is generated. This list is non-editable and is used for selecting warehouse attributes and composing warehouse conditions. The default operational metadata file name is "demo" in the current prototype and cannot process files with other file names. If the typed in operational

metadata file name does not exist in the data files, error information will be presented in this region [1]. Limiting the file naming convention and placing the file in the precise application folder for the system to be able to process is a design concern that is intended to address. If the file is placed in a different location other than the root folder of the Java DWQTI application folder, the application would generate an error. User needs to generate total 3 metadata files - demo_re.txt, demo_fd.txt, demo_jd.txt.

3.2.2. Warehouse attributes

This section allows user to select warehouse attributes corresponding to the SELECT part of an operational database query in SQL format. All the attributes listed in this region are selected from the "Operational DB Attributes" list. Clicking on an attribute in the list of operational database attributes will place it in the list of warehouse attributes in this region, and double clicking on an attribute in the list of operational database attributes will remove it from the list of warehouse attributes. Once the desired attributes are selected the user can click on the "Fix Warehouse Attributes" button to save the attributes. Clicking on the button "Reset" would just clear the selected list of warehouse attributes [1].

3.2.3. Warehouse conditions

This region is an editable text area for users to compose a warehouse condition. The warehouse condition is of the form C_1 & C_2 & ….. & C_m, where each C_i is a simple condition or the union of a few simple conditions. A simple condition can be - *`selected attribute' op ` selected attribute or constant'*, where op is one of the six comparison operators: =, >, <, <=, >=, and !=. For example, A = "Ames" & B <=120 | B >= 400 & C < E. Users can click on the button labeled "Legal Conditions" to verify if the warehouse condition composed is legal. The small text field under this button will display either "Yes!" or some error message.

Any attribute in a warehouse condition can be either typed in or placed by selecting the relevant attribute in the list of operational DB attributes and then clicking on the button labeled with "Get One Attribute". Clicking on some operator in the list of "Operators" will place it to the current cursor position in the text area. All constants (either real numbers or strings) are typed in. Clicking on the button labeled with "Clear" clears the text area [1]. The step to define warehouse conditions involves several steps by user before the condition can be formed and this has been another design enhancement to be addressed.

3.2.4. Operational DB query

This is a non-editable text area for displaying the operational database query in SQL. After clicking on the button labeled with "Get Operational DB Query", users will see either the desired operational database query in SQL format (i.e. SELECT, FROM, WHERE) corresponding to the given warehouse attributes and conditions, or the information of "A lossy join is made!" together with the computed optimal join sequence for user's reference purpose. The join sequence can then be used to manually derive an SQL [1].

3.2.5 Examples of the operational DB query

Example 1:

Operational Meta Data File: demo

Warehouse Attributes: dname, ename, iname

Warehouse Condition: inum = 12 & mbirth <= "9/9/54" | mbirth >= "10/22/60" & ocity = "Ames"

Operational DB Query:

SELECT dept.dname, employee.ename, item.iname

FROM owner, dept, employee, manager, store_info, sale, item

WHERE sale.inum = 12 AND (manager.mbirth <= "9/9/54" OR manager.mbirth >= "10/22/60") AND owner.ocity = "Ames" AND owner.storenum = dept.storenum AND owner.onum = store_info.onum AND owner.storenum = store_info.storenum AND owner.storenum = sale.storenum AND dept.mnum = employee.mnum AND dept.mnum = manager.mnum AND dept.dnum = sale.dnum AND employee.enum = sale.enum AND sale.inum = item.inum

For this example, the warehouse or input query consists of two parts: the warehouse attributes *dname*, *ename*, and *iname*, and the warehouse condition. In the output query (or) the operational database query in SQL format, the SELECT part corresponds to the three warehouse attributes, the FROM part tells us the join sequence, and the WHERE part presents the necessary conditions for the query. Among the conditions in the WHERE part, the first two correspond to the warehouse condition, and the others correspond to the join sequence so that a lossless join can be performed.

Example 2:

Operational Meta Data File: demo

Warehouse Attributes: dname, iname, lname, mname, pname

Warehouse Condition: isnum >= inum

Operational DB Query:

Optimal Join Path: lawyer - dept - manager - store_info - sale - item - parts

A lossy join is made.

In this example, the input query (or the warehouse query) consists of five warehouse attributes and the warehouse condition. The output query presents the join sequence and a message "A lossy join is made" for user's reference purpose. The join sequence can then be used to manually derive an SQL. The issue is that the user will understand that the join as given will

need to be considered lossy and also the design of the warehouse will need to be reconsidered or the join sequence will have to be expanded [1].

3.3. Interface implementation

The DWQTI is implemented in Java (JDK 1.0.2). The Java program is a standalone application implemented as a public class (named DWQTI) which extends the Java AWT class Frame. The class DWQTI consists of the main( ) method, fourteen new-designed methods, and additional methods inherited from the AWT class Frame. Also, seven utility classes and six data structure classes are designed and implemented to support the fourteen methods. Table 1 shows the classes used in the Java program [1].

Class	Objective
DWQTI	Provide user interface functions
FindJoinPath	A utility class to find a join sequence
LosslessTest	A utility class to test the losslessness of the join
RelationOfTwoSets	A utility class to determine containership
CharCount	A utility class to count character strings
LegalOrNot	A utility class to determine legality of operator
SortDelVector	A utility to sort
ReadmeWin	A utility class to display readme
CondNode	Data structure class to hold condition
RelNode	Data structure class to hold the relation schema
ABFSNode	Data structure class for the ABFS tree
CIGNode	Data structure class to hold the intersection graph
FDNode	Data structure class for holding functional dependency
ASetNode	Data structure class for holding the Adjustment set

Table 1: Classes used in the CASE tool [1]

3.5. Modeler foundations and implementation defined

3.5.1. Query translation process

Here is an explanation of the query translation process implemented in the legacy application. To create a warehouse query, the query translation process involves three main steps [1]:

1 Translate the request for the warehouse design into hypergraph space.

2 The resulting source query (warehouse) then be mapped to the target data space (the hypergraph representing the collection of connected operational databases)

3 Finally, the target query hypergraph is mapped to an SQL query.

The basic data structures and algorithms such as hypergraph notion and its complete intersection graph, adjusted breadth first search are applied to generate the DB query. Finding an optimal join sequence for the selected warehouse attributes, including the attributes appeared in the warehouse condition is a crucial part in the modeler design and implementation. After a join sequence for the target hypergraph is found, the modeler implements the lossless test of the join sequence. The lossless test of a join sequence is basically to determine if a sub-hypergraph defines an embedded join dependency (EJD) [1].

As mentioned in Section 3.3, the implementation of this modeler or engine part for the DWQTI Java program includes two utility classes and four data structure classes. The functionality and relations of these six classes and their methods are shown in Tables 2 and 3. The implementation of this modeler also extensively uses the following utility classes and data structure classes introduced in Section 3.3: class SortDelVector, class CharCount, class RelationOfTwoSets, and class RelNode.

Method	Objective
FindJoinPath( )	Constructor
makeCIG( )	Create a vector of CIG nodes
isLegalRoot( )	Verifies the legality of a given root node
canBeAdded( )	Checks if node can be added to search tree
resetASet	Removes all ASet nodes
makeABFSTree( )	Creates the ABFS tree
findJoinPath( )	Finds a join sequence
nodeLevel( )	Computes the level of a node
markNodes( )	Marks a leaf and its ancestors
hasDiffAttr( )	Checks for warehouse attribute

Table 2: Methods of Utility class FindJoinPath

Method	Objective
LosslessTest( )	Constructor
readFDs( )	Reads functional dependencies from metafile
readJDs( )	Reads join dependencies from metafile
hasCommonNonWAttr( )	Checks for common non warehouse attribute
isLossless( )	Tests a join sequence for losslessness
byChaseAlgorithm( )	Alternate way to test losslessness

Table 3: Methods of Utility class LosslessTest

4. New Query Translation Interface Design and Implementation - .Net Application

4.1. Query Translation Interface overview

This section presents the refactored model of DWQTI interface using C# language under.Net framework. As user friendliness was a significant design issue in the current legacy tool; the main goal during refactoring the application is to build a simplified and streamlined UI keeping

the existing functionality and modeler system design same as the old legacy system. A change was made to the internal structure of software to make it easier to understand and modifying without changing its observable behavior. As seen, the legacy java program code has several classes and methods and has a very sophisticated workflow. With the emerging version of new software, extending or modifying the legacy application is infeasible and the need for transformation into newer software has evolved. The code refactoring covers optimizations of code and improves some nonfunctional attributes of the software using inbuilt classes in .Net. Advantages include improved code readability and reduced complexity to improve the maintainability of the source code, as well as a more expressive internal architecture or object model to improve extensibility. The end result is an application supported on latest windows operating systems that preserves existing functionality and allows future development and improvements as required. Section 4.2 explains about the refactoring approach. In Section 4.3, a general description of the new DWQTI interface design is presented. In Section 4.5, the implementation of the user interface is discussed. Section 5 covers the future work.

4.2. Refactoring approach

Several research papers that are listed in the references section have been referred to understand the basic data structures and algorithms necessary to do the query translation. Basic concepts of relational database theory, Chase method with tableaux processing, concepts and properties of the hypergraph model, acyclicity and hinge have been studied. Also, the code analysis is done on legacy application by debugging code to identify the application workflow.

Refactoring of legacy applications comes with many challenges and several lessons to be learned during the process. With lack of conversion tools readily available to automate the java code and data conversion to .Net; the approach to rewrite the code in .Net has been implemented.

Also, the method to assess the nature of the system (or) code analysis was done through code debugging. Debugging the legacy code and reading proved that it is both time consuming and confusing. Hence mindmaps and UML diagrams are used to sketch down the workflow and class relationships while performing code analysis. A snapshot of the mind map created is shown in the figure 3.

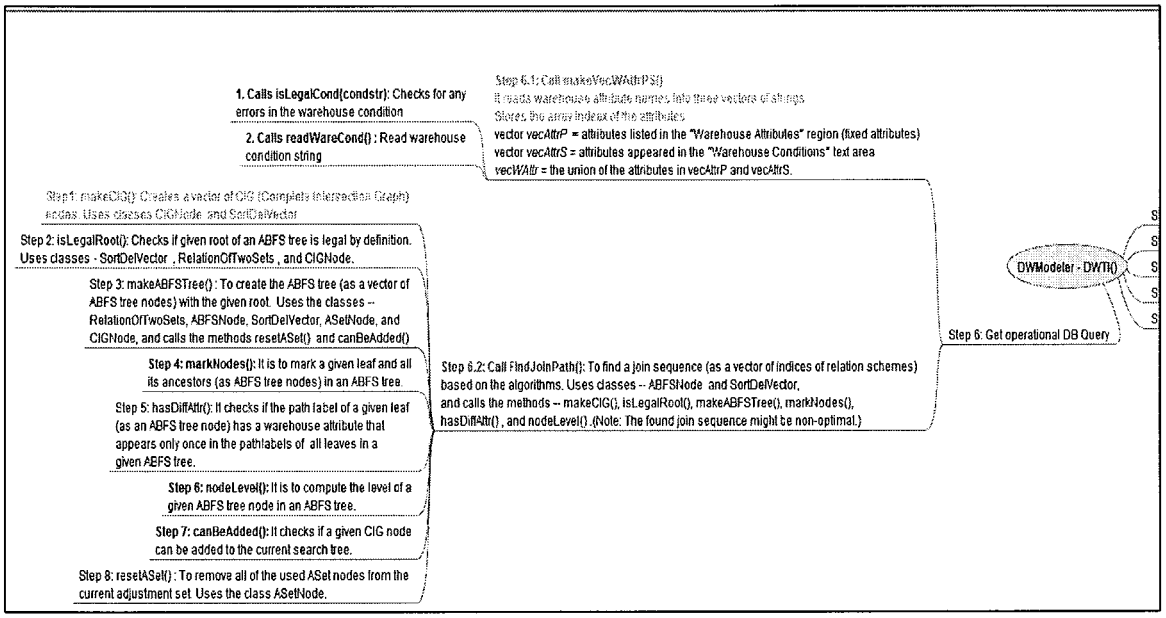

Figure 3: Mind map of the legacy application workflow

Preserving behavior was another large challenge while rewriting the application; hence an iterative and incremental development approach and continuous feedback from the stakeholders is taken. The book "Working effectively with legacy code" [8] provides useful guidance and describes a series of practical strategies that developers can employ when refactoring or making functional changes to legacy codebases. Some of the strategies employed here are the use of mind maps, and as well the iterative and incremental development approach,

4.3. Process improvements

The new prototype of the DWQTI CASE tool along with the existing functionality also includes additional features to manage the metadata files and a search capability. The approach was to enhance its internal structure without changing its external behavior. Figure 3 displays the improvised UI of the query translation tool with better navigation for users.

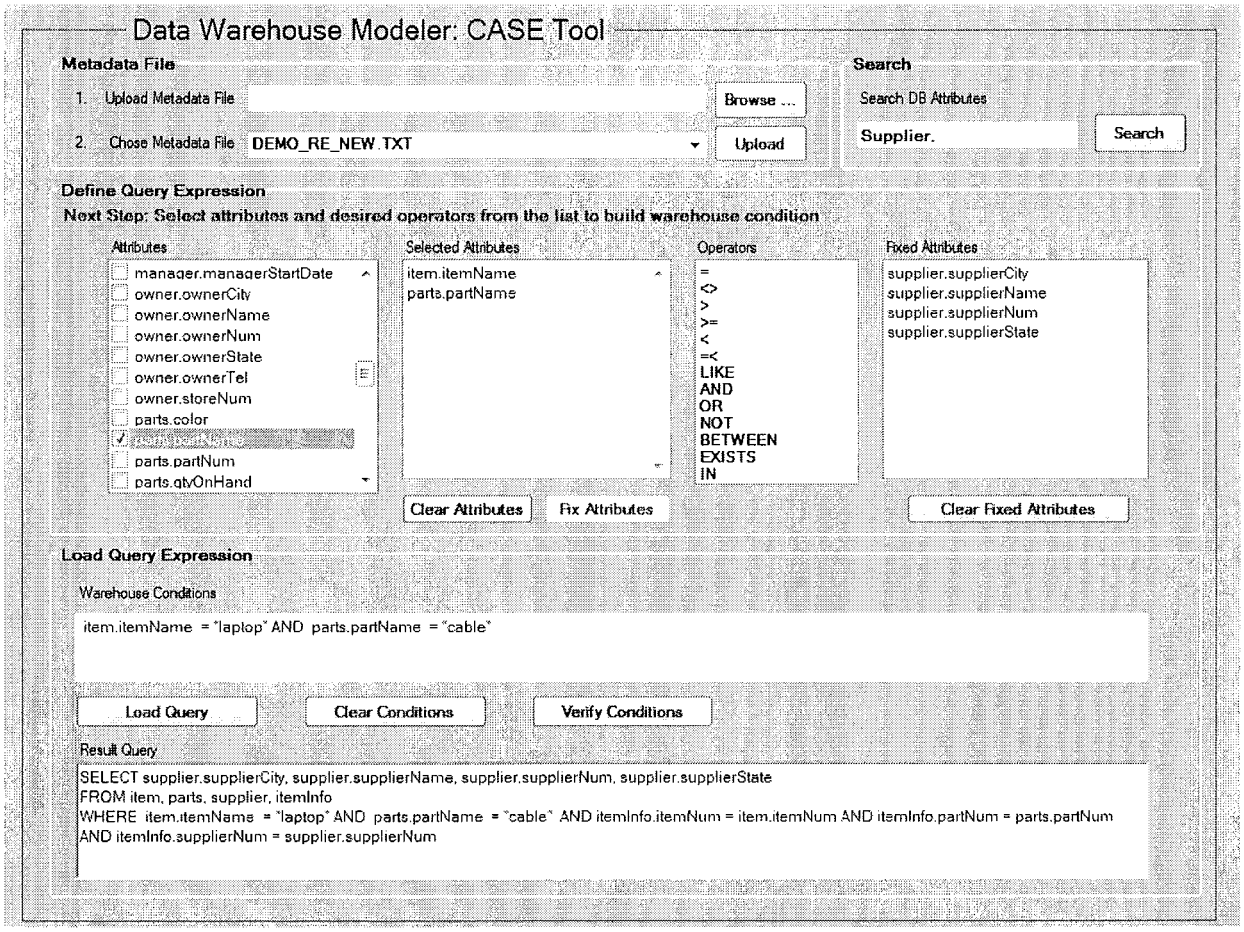

Figure 4: New Data Warehouse Query Translation Interface – .Net Windows Application

4.2.1. Upload metadata file

Unlike the legacy system, the user can now browse for the metadata file locally and upload it to the system for processing. Now, users don't have to have an exact filename for the metadata

file. The new system can process metadata file of any filename as long as the file data has a valid format. All uploaded files are stored locally to the application folder and can be referenced for future use. If the user uploads two files with a similar file name, a confirmation to overwrite the existing file with the new file is displayed to the user. User can either chose to overwrite the existing file or upload the file with a different name.

4.2.2. Chose metadata file

All the uploaded files are stored locally to the application folder. User can chose the relevant metadata file from the drop-down list and click on the "Upload" button. A scrolling list of all attribute names in the given operational metadata is generated and displayed under list box labeled "Attributes". As in legacy application, this list is non-editable and is used for selecting warehouse attributes and composing warehouse conditions. Upon any errors in processing the metadata file, an error message – "The <<file name>> cannot be processed. Please check the format" is displayed. User can either chose to correct and re-upload the file with same file name (or) upload a new metadata file for processing.

4.2.3. Search DB attributes

A new capability to search for the DB attributes has been added to the interface. By default, the search option is disabled until the user uploads the metadata file and a scrolling list of attributes is generated by the application. Once enabled, the user can perform a wild search on the table and column names. The search results will then be added automatically to the list box labeled "Selected Attributes". User can then either choose to add or delete the columns from the list. If no search results are generated by the system, a message – "No search results returned!" is prompted to the user.

4.2.4. Fixed attributes

"Define Query Expression" group allows user to select warehouse attributes corresponding to the SELECT part and Warehouse condition of an operational database query. All the attributes listed in the "Attributes" list box are generated from the metadata file that user has uploaded. Clicking on an attribute name in the list of operational database attributes will place it in the list of warehouse attributes under "Select Attributes" column; and a click again on an attribute in the list of operational database attributes will remove it from the list of selected attributes. Once the desired attributes are selected the user can click on the "Fix Warehouse Attributes" button to fix the attributes for the SELECT part of the operational database query. Clicking on the button "Clear Attributes" will reset the selected attributes column. A label on the top of the list instructs the user on the next desired action on the application. Adding attributes to "Fix Attributes" column is optional and if no columns are selected, the system will add all columns of the selected tables to the SELECT part.

4.2.5. Warehouse conditions

This region is an editable text area for users to compose a warehouse condition. Similar to selecting the attributes for the SELECT part, user can perform the related actions to select attributes to build the warehouse condition. Any attribute in a warehouse condition can be either typed in or placed by selecting the relevant attribute in the list of selected DB attributes and then by clicking on some operator in the list of "Operators". This will place the attribute and the operator at the current cursor position in the text area of "Warehouse Conditions". All constants (either real numbers or strings) are typed in. Clicking on the button labeled with "Clear Conditions" clears the text area.

Unlike the legacy system, the new interface now provides more comparison operators to build the warehouse condition, whereas the formation of the warehouse condition is same as the legacy application i.e. - *`selected attribute' op ` selected attribute or constant'*, where op is one of the comparison operators in the given list. For example, A = "Ames" AND B <=120 OR B >= 400 AND C < E. Users can click on the button labeled "Verify Conditions" to verify if the warehouse condition composed is legal. A message box with relevant error message is displayed on the screen, if invalid condition (or) "No Errors Found. Valid Condition!" is shown upon no errors. Also, the new interface has more verification applied to validate the warehouse condition for errors compared to the java application.

4.2.6. Load query expression

This is a non-editable text area for displaying the operational database query in SQL. After clicking on the button labeled "Load query", users will see either the desired operational database query in SQL format (i.e. SELECT, FROM, WHERE) corresponding to the given warehouse attributes and conditions, or the information of "A lossy join is made!". The fundamentals and internal logic of query generation is applied same as the legacy application.

4.3. User guide and tool tips

Labels and tool tips have been added to the interface to guide user on the next steps required on the UI and provide information about the UI buttons and tools provided.

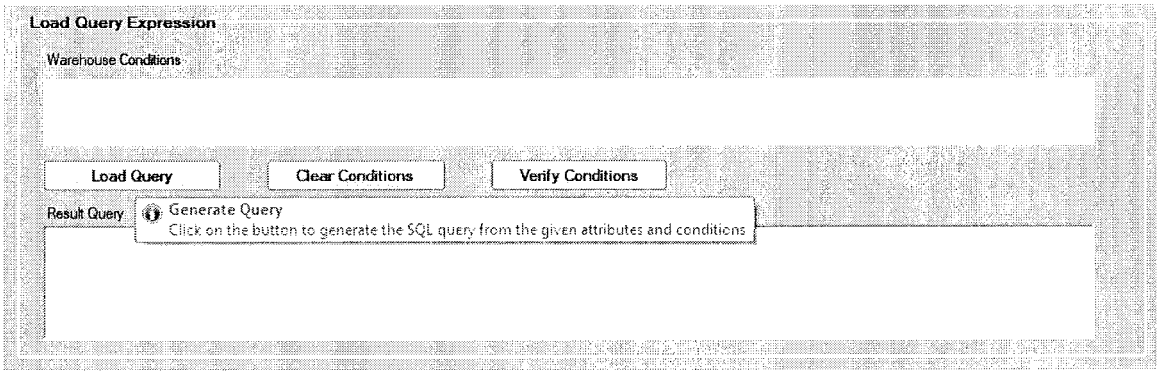

Figure 5: Display of tool tips on the user interface

4.5. Interface implementation

The new DWQTI is implemented on MS .Net 4 using C# language. It's a standalone windows application implemented as a public class named QueryInterface that extends the QueryInt class. The QueryIntclass consists fourteen new-designed methods in total that implement that's refactored from Java application. Table 4 shows the classes and methods of the .Net program and Figure 5 shows the class diagram.

Method	Objective
btnBrowseInput_Click()	Browse for the metadata files
btnFixAttr_Click()	Handles button click event of fix attributes
btnResetFixAttribs_Click()	Handles button click event of reset fix attributes
ClearAttributes_Click()	Handles button click event of clear attributes
columnNames_ItemCheck()	Select attributes from the attributes list
searchAttribsBtn_Click( )	Search for attributes. Calls the method FindAllOfMyString()
FindAllOfMyString( )	Looks for the keyword entered for search in the warehouse attributes list
getRelationships( )	Process the Demo_FD file and store all the table relations in an array
listShowAttributes_DoubleClick()	Handles the click event on the columns added to the

	select attributes list
operators_DoubleClick()	Handles the click event on the operator symbol given in the operators listbox
QueryInt()	
QueryInt_Load()	Reads the metadata file and adds data to the attributes list
resetConditions_Click()	Handles button click event of reset conditions
showQueryResult_Click()	Handles button click event of load query event. Class method validate() and getRelationships() to generate the SQL query
upload_Click()	Process the metadata file and save the data in array
verifyConditions_Click()	Calls the Validate() method that verifies the WH condition for errors
Validate()	Verify the warehouse condition for errors

Table 4: Methods of QueryInt class

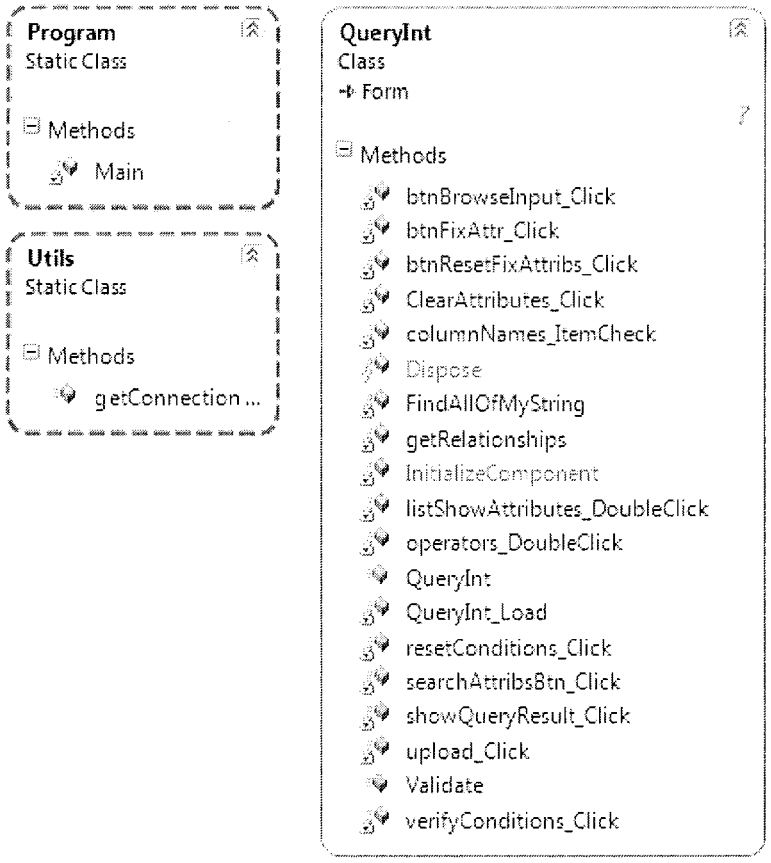

Figure 6: Query Interface Class Diagram

4.5.1 Metadata files

The metadata files used – demo_re.txt and demo_fd.txt, has been also refactored for better readability and understanding of the table and column names; whereas, in order to test results of Java and .Net problem; the DB structure and column relationships were retained similar to the old metadata files. And, Demo_JD file is merged with Demo_FD file in the new application.

File Format - DEMO_RE.TXT

supplier:supplierNum,supplierName,supplierCity,supplierState;

supply: itemNum,supplyNum,partNum,quantity,shipDate;

itemInfo:itemNum,partNum,supplierNum;

item:itemNum,itemName,itemPrice,itemQtyOnHand,supplierNum;

parts:partNum,partName,color,weight,qtyOnHand,supplierNum;

sale:itemNum,itemSalenum,itemSaledate,itemSaleQty,empNum,deptNum,storeNum;

partSale:partNum,partSaleNum,partSaleDate,partSaleQuan,empNum,deptNum,storeNum;

store:storeNum,storeState,storeCity;

owner:storeNum,ownerNum,ownerName,ownerTel,ownerCity,ownerState;

lawyer:storeNum,lawyerNum,lawyerName,lawyerTel,lawyerCity,lawyerState;

storeInfo:storeNum,ownerNum,lawyerNum;

employee:empNum,empName,empSalary,empStartDate,empBirth,managerNum;

dept:deptNum,deptName,floor,managerNum,storeNum;

manager:managerNum,managerName,managerSalary,managerStartDate,managerBirth;

File Format - DEMO_FD.TXT

supplierNum->supplierName,supplierCity,supplierState;

itemNum->itemName,itemPrice,supplierNum;

partNum->partName,color,weight,supplierNum;

itemSalenum->itemSaledate,itemNum,storeNum;

partSaleNum->partSaleDate,partNum,storeNum;

storeNum->storeState,storeCity;

storeNum->ownerNum,lawyerNum;

ownerNum->ownerName,ownerTel,ownerCity,ownerState;

lawyerNum->lawyerName,lawyerTel,lawyerCity,lawyerState;

empNum->empName,empStartDate,empBirth,empSalary,managerNum;

deptNum->deptName,managerNum,storeNum;

managerNum->managerName,managerStartDate,managerBirth,managerSalary;

5. Conclusion and future work

Decision support systems not only need a data warehouse as a central repository, but also front-end techniques that facilitate accessing and retrieving information. Fast access to a data warehouse plays a major role for decision makers. Also, the growing significance of the user interface which is critical for effective system usability and performance; incorporating guidelines for designing user interface software has been a better software development practice [10]. Hence, the need to refactor the original java application to .Net has emerged.

The new Query Translation Interface is designed strongly based on the results in [4] [3] [5] and is implemented in .Net Framework 4.0 using language C#. It provides an improved UI and an efficient modeler to let the users obtain the desired operational database query information in SQL. The .Net program has been run and tested on Windows 7.0 platform and developed in Visual Studio 2010. The data structure classes and utility classes in this program are well designed so that the implementation is relatively simple and efficient compare to the legacy program.

As a future work, the program is subjected to be tested by real-time users and conduct a survey on the efficiency of the tool. As an another possible extension of this program, an ability to save the warehouse query (input) and the resulting operational database query (output) into an external file can be added for further reference. Also, it is possible to connect this user interface to the underlying operational databases using the database connectivity in .Net; the resulting operational database query can be run directly on operational databases and retrieve the desired

query results from the databases and display on a data grid. Finally, a comparative study of the original DWTI versus the new improved system can be performed.

References

1. Wu, L., Miller, L., & Nilakanta, S. (2001). Design of data warehouses using metadata. Information and Software Technology, 43(2), 109-119.
2. Miller, L.; Nilakanta, S., "Data Warehouse Modeler: a CASE tool for warehouse design," System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on , vol.6, no., pp.42,48 vol.6, 6-9 Jan 1998
   doi: 10.1109/HICSS.1998.654756
3. Owrang O, M. M., & Miller, L. L. (1988). Query translation based on hypergraph models. The Computer Journal, 31(2), 155-164.
4. Miller, L. L., Leuchner, J. H., Kothari, S., & Liu, K. C. (1990). Testing arbitrary subhypergraphs for the lossless join property. Information Sciences, 51(1), 95-110.
5. Lin, H. N., Miller, L. L., & Owrang O, M. M. (1991, April). A universal relation language interface for INGRES. In Proceedings of the 19th annual conference on Computer Science (pp. 321-331). ACM.
6. Bontempo, C., & Zagelow, G. (1998). The IBM data warehouse architecture. *Communications of the ACM*, *41*(9), 38-48.
7. Ponniah, P. (2004). *Data warehousing fundamentals: a comprehensive guide for IT professionals*. Wiley. Com
8. Feathers, M. (2004). *Working effectively with legacy code*. Prentice Hall Professional.

9. Douglas, S., Tremaine, M., Leventhal, L., Wills, C. E., & Manaris, B. (2002). Incorporating human-computer interaction into the undergraduate computer science curriculum. *ACM SIGCSE Bulletin, 34*(1), 211-212.
10. Smith, S. L., & Mosier, J. N. (1986). *Guidelines for designing user interface software.* Bedford, MA: Mitre Corporation.

List of Figures

List of Tables

6. Appendix

A. Data Files

File Format - DEMO_RE.TXT

supplier:supplierNum,supplierName,supplierCity,supplierState;

supply:itemNum,supplyNum,partNum,quantity,shipDate;

itemInfo:itemNum,partNum,supplierNum;

item:itemNum,itemName,itemPrice,itemQtyOnHand,supplierNum;

parts:partNum,partName,color,weight,qtyOnHand,supplierNum;

sale:itemNum,itemSalenum,itemSaledate,itemSaleQty,empNum,deptNum,storeNum;

partSale:partNum,partSaleNum,partSaleDate,partSaleQuan,empNum,deptNum,storeNum;

store:storeNum,storeState,storeCity;

owner:storeNum,ownerNum,ownerName,ownerTel,ownerCity,ownerState;

lawyer:storeNum,lawyerNum,lawyerName,lawyerTel,lawyerCity,lawyerState;

storeInfo:storeNum,ownerNum,lawyerNum;

employee:empNum,empName,empSalary,empStartDate,empBirth,managerNum;

dept:deptNum,deptName,floor,managerNum,storeNum;

manager:managerNum,managerName,managerSalary,managerStartDate,managerBirth;

File Format - DEMO_FD.TXT

supplierNum->supplierName,supplierCity,supplierState;

itemNum->itemName,itemPrice,supplierNum;

partNum->partName,color,weight,supplierNum;

itemSalenum->itemSaledate,itemNum,storeNum;

partSaleNum->partSaleDate,partNum,storeNum;

storeNum->storeState,storeCity;

storeNum->ownerNum,lawyerNum;

ownerNum->ownerName,ownerTel,ownerCity,ownerState;

lawyerNum->lawyerName,lawyerTel,lawyerCity,lawyerState;

empNum->empName,empStartDate,empBirth,empSalary,managerNum;

deptNum->deptName,managerNum,storeNum;

managerNum->managerName,managerStartDate,managerBirth,managerSalary;

B. Source Code

program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace QueryInterface
{
  static class Program
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new QueryInt());
    }
```

```
    }
}
```

QueryInt.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace QueryInterface
{
    public partial class QueryInt : Form
    {
    #region "Global Variables"
        List<WHConditions> WHC = new List<WHConditions>();
        List<RelationTableData> Tables = new List<RelationTableData>();
```

```
    List<EnityDefinitions> PrimaryKeys = new List<EnityDefinitions>();
    string selectedTables = string.Empty;
    DataSet ds = new DataSet();
    List<string> tabs = new List<string>();
    List<string> ExtraWHTabs = new List<string>();
#endregion

#region "Initialize Component"
    public QueryInt()
    {
        InitializeComponent();
    }
#endregion

#region "Get Files and Add to the databaseNames Items List"
    private void QueryInt_Load(object sender, EventArgs e)
    {
        try
        {
            string filePath = Path.Combine(Environment.CurrentDirectory, @"..\..\..\");
            string[] fileEntries = Directory.GetFiles(@filePath, "*.txt");
            databaseNames.Items.Insert(0, "Select");
```

```
            int i = 1;
            //Get list of text files under the folder and add to the databaseNames list item
            foreach (string fileName in fileEntries)
            {
               string sOnlyFilename = System.IO.Path.GetFileName(fileName);
               databaseNames.Items.Insert(i, sOnlyFilename);
               i++;
            }
            databaseNames.SelectedIndex = 0;
         }
         catch (Exception ex)
         {
            MessageBox.Show("Failed to get schema file", ex.ToString());
         }
      }
#endregion

#region "Process the text file and save data in array"
   private void upload_Click(object sender, EventArgs e)
   {
      if (!(databaseNames.SelectedIndex <= 0))
      {
```

```
Tables.Clear();
grpSearch.Enabled = true;
columnNames.Items.Clear();
ds.Tables.Clear();
string selectedFile = databaseNames.SelectedItem.ToString();
FileInfo file = new FileInfo(selectedFile);
DataTable dt = null;
DataColumn dc = null;
string filePath = Path.Combine(Environment.CurrentDirectory, @"..\..\..\" + selectedFile);
string[] lines = File.ReadAllLines(@filePath);
if (lines.Length > 0)
{
   foreach (string line in lines)
   {
      if (line.IndexOf(";") > -1)
      {
         string[] tlines = line.Split(";".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
         foreach (string tline in tlines)
         {
            if (tline.Length > 0 && tline.IndexOf(":") > -1)
            {
               string[] items = tline.Split(":".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
```

```
                dt = null;
                dt = new DataTable(items[0]);
                tabs.Add(items[0]);
                string[] cols = items[1].Split(new string[] { ",", ";" }, StringSplitOptions.RemoveEmptyEntries);
                Tables.Add(new RelationTableData(items[0], cols.ToList()));
                foreach (string col in cols)
                {
                  if (col.Length > 0)
                  {
                    dc = new DataColumn(col.Trim());
                    dt.Columns.Add(dc);
                    columnNames.Items.Add((dt.TableName.Trim() + "." + dc.ColumnName), false);
                    dc = null;
                  }
                }
                columnNames.Sorted = true;
                if (!ds.Tables.Contains(dt.TableName))
                  ds.Tables.Add(dt);
              }
            }
          }
        }
```

```
            getRelationships();
        }
        else
            grpSearch.Enabled = false;

        //Check to see if the file has been uploaded.
        if (columnNames.Items.Count == 0)
        {
            MessageBox.Show("The file " + selectedFile + " cannot be processed. Please check the format");
            grpSearch.Enabled = false;
        }
        else
            lblMessage.Text = "Next Step: Select the required attributes from the list and click on 'Fix Attributes' button";
    }
}
#endregion

#region "Select Attributes from the list"
    private void columnNames_ItemCheck(object sender, ItemCheckEventArgs e)
    {
        int i = columnNames.SelectedIndex;
        string selectedColumn = columnNames.SelectedItem.ToString();
```

```
        if ((listShowAttributes.Items.IndexOf(selectedColumn) > -1))
            listShowAttributes.Items.Remove(selectedColumn);
        else
            listShowAttributes.Items.Add(selectedColumn);
    }
#endregion

#region "Clear Attributes from the list"
    private void ClearAttributes_Click(object sender, EventArgs e)
    {
        for (int i = 0; i < columnNames.Items.Count; ++i)
            columnNames.SetItemChecked(i, false);
        listShowAttributes.Items.Clear();
    }
#endregion

#region "Show Query Result"
    private void showQueryResult_Click(object sender, EventArgs e)
    {
        string selectedAttributes = string.Empty;
        string selectedTables = string.Empty;
```

```
string relations = string.Empty;
string conditions = string.Empty;
List<string> sd = new List<string>();
List<string> Query = new List<string>();
IEnumerable<string> flds = new string[] { };
int cnt = 0;
bool SingleTable = false;
List<string> sd1 = new List<string>();

//Checks the verify condition again
if (!Validate())
{
#region Writes Selected items and Selected Tables from Fixed Variable
   if (lstFixedAttributes.Items.Count > 0)
      foreach (object item in lstFixedAttributes.Items)
      {
         selectedAttributes = (selectedAttributes.Length > 0 ? (selectedAttributes + ", " + item.ToString()) : item.ToString());
         sd.Add(item.ToString());
      }
   else
      selectedAttributes = "*";
   sd = sd.Select(w => w.ToString().Split('.').ToArray()[0]).Distinct().ToList();
```

```
    selectedAttributes = selectedAttributes + Environment.NewLine + "FROM ";
#endregion

#region Selects the Where Conditions and its Related Tables
    List<string> Wtavle = WHC.Select(w => w.AttrName.Split('.').ToArray()[0]).Distinct().ToList();
    Wtavle.AddRange(sd); Wtavle.AddRange(ExtraWHTabs); Wtavle = Wtavle.Distinct().ToList(); ;
    if (Wtavle.Count > 0)
        foreach (string item in Wtavle)
            selectedTables = (selectedTables.Length > 0 ? (selectedTables + ", " + item.ToString()) : item.ToString());
#endregion

#region Operating DB Query
    if (Wtavle.Count == 1)
        SingleTable = true; // if single table selected
    else
    {
        for (int i = 0; i < Wtavle.Count - 1; i++)
            for (int j = i + 1; j < Wtavle.Count; j++)
            {
                for (int relCnt = 0; relCnt < Tables.Count; relCnt++)
                {
```

```
string[] PKs = new string[] { PrimaryKeys.Where(p => p.FieldName.Equals(Wtavle[i])).FirstOrDefault().TableName,
PrimaryKeys.Where(p => p.FieldName.Equals(Wtavle[j])).FirstOrDefault().TableName };
HashSet<string> tbl = new HashSet<string>(Tables[relCnt].TableName);
flds = tbl.Intersect(PKs);
if (flds.Count() > 1)
{
  string Table1 = PrimaryKeys.Where(p => p.TableName.Equals(PKs[0])).FirstOrDefault().FieldName;
  string Table2 = PrimaryKeys.Where(p => p.TableName.Equals(PKs[1])).FirstOrDefault().FieldName;
  string field1 = PrimaryKeys.Where(p => p.FieldName.Equals(Table1)).FirstOrDefault().TableName;
  string field2 = PrimaryKeys.Where(p => p.FieldName.Equals(Table2)).FirstOrDefault().TableName;
  if (!Wtavle.Contains(Tables[relCnt].FieldName)) sd1.Add(Tables[relCnt].FieldName); ;

  if (!Tables[relCnt].FieldName.Equals(Table1) && !Tables[relCnt].FieldName.Equals(Table2))
  {
    string s1 = Tables[relCnt].FieldName + "." + field1 + " = " + Table1 + "." + field1;
    if (!(Query.Where(q => q.Contains(s1)).Count() > 0)) Query.Add(s1);
    string s2 = Tables[relCnt].FieldName + "." + field2 + " = " + Table2 + "." + field2;
    if (!(Query.Where(q => q.Contains(s2)).Count() > 0)) Query.Add(s2);
  }
  else if (Tables[relCnt].FieldName.Equals(Table1) && !Tables[relCnt].FieldName.Equals(Table2))
  {
    string s = Tables[relCnt].FieldName + "." + field2 + " = " + Table2 + "." + field2;
```

```
                    if (!(Query.Where(q => q.Contains(s)).Count() > 0)) Query.Add(s);
                }
                else if (!Tables[relCnt].FieldName.Equals(Table1) && Tables[relCnt].FieldName.Equals(Table2))
                {
                    string s = Tables[relCnt].FieldName + "." + field1 + " = " + Table1 + "." + field1;
                    if (!(Query.Where(q => q.Contains(s)).Count() > 0)) Query.Add(s);
                }
                cnt++;
                break;
            }
        }
    }
}
#endregion

string WHCText = conditionsEntry.Text;
if (!string.IsNullOrEmpty(WHCText)) WHCText = WHCText + " AND ";
StringBuilder QryText = new StringBuilder();
StringBuilder SelTab = new StringBuilder();
for (int i = 0; i < Query.Count; i++)
{
   QryText.Append(" AND ");
```

```
            QryText.Append(Query[i]);
        }
        sd1 = sd1.Distinct().ToList();

        for (int sdcnt = 0; sdcnt < sd1.Count; sdcnt++)
        {
            if (!string.IsNullOrEmpty(SelTab.ToString()))
                SelTab.Append(", ");
            SelTab.Append(sd1[sdcnt]);
        }

        selectedTables = selectedTables + ((!string.IsNullOrEmpty(SelTab.ToString())) ? (", " + SelTab.ToString()) : "") +
Environment.NewLine + "WHERE ";
        if (cnt >= Wtavle.Count || (flds.Count() > 1 && cnt == 1) || SingleTable)
            txtResult.Text = "SELECT " + selectedAttributes + selectedTables + conditionsEntry.Text + QryText.ToString();
        else
            MessageBox.Show("Loosely Joined");
      }
    }
  #endregion

  #region "reset conditions"
```

```
    private void resetConditions_Click(object sender, EventArgs e)
    {
        conditionsEntry.Text = "";
        if (conditionsEntry.Text == string.Empty)
            txtResult.Text = string.Empty;
    }
#endregion

#region "Browse for the metadata files"
    private void btnBrowseInput_Click(object sender, EventArgs e)
    {
        DialogResult dr = this.openFileDialogText.ShowDialog();
        if (dr != System.Windows.Forms.DialogResult.Cancel)
        {
            // Show the file path
            dbFilePath.Text = openFileDialogText.FileName;
            int currentIndex = databaseNames.Items.Count;

            //Read the file name
            string inFileName = openFileDialogText.SafeFileName;
            databaseNames.Items.Insert(currentIndex, inFileName.ToUpper());
            databaseNames.SelectedIndex = currentIndex;
```

```
string activeDir = Path.Combine(Environment.CurrentDirectory, @"..\..\..\");
string newPath = System.IO.Path.Combine(activeDir, inFileName.ToUpper());
if (!System.IO.File.Exists(newPath))
{
  try
  {
    File.Copy(dbFilePath.Text, newPath);
    MessageBox.Show("File upload complete!");
  }
  catch (Exception ex)
  {
    MessageBox.Show("Failed to upload the file", ex.ToString());
  }
}
else
{
  string message = "There is already a file with the same name in this location. Do you want to overwrite the file";
  string caption = "Copy File";
  MessageBoxButtons buttons = MessageBoxButtons.YesNo;
  DialogResult result;

  // Displays the MessageBox.
```

```
            result = MessageBox.Show(this, message, caption, buttons,
            MessageBoxIcon.Question, MessageBoxDefaultButton.Button1);

            if (result == DialogResult.Yes)
            {
              try
              {
                File.Copy(dbFilePath.Text, newPath, true);
                MessageBox.Show("File upload complete!");
              }
              catch (Exception ex)
              {
                MessageBox.Show("Failed to upload the file", ex.ToString());
              }
            }

          }
        }
      }
#endregion

#region Get Relations
```

```
private string getRelationships()
{
   string retValue = string.Empty;
   string filePath = Path.Combine(Environment.CurrentDirectory, @"..\..\..\DEMO_FD_NEW.txt");
   string[] lines = File.ReadAllLines(@filePath);
   if (lines.Length > 0)
   {
      foreach (string line in lines)
      {
         if (line.IndexOf(";") > -1)
         {
            string tlines = line.Split("->".ToCharArray(), StringSplitOptions.RemoveEmptyEntries)[0];
            string[] fields = line.Split(new string[] { "->", ";" }, StringSplitOptions.RemoveEmptyEntries)[1].Split(',');
            foreach (RelationTableData rd in Tables)
            {
               HashSet<string> st = new HashSet<string>(rd.TableName);
               IEnumerable<string> fie = st.Intersect(fields);
               if (fie.Count() >= 2)
                  PrimaryKeys.Add(new EnityDefinitions(tlines, rd.FieldName));
            }
         }
      }
```

```
        }
        return retValue;
    }
#endregion

#region Click Events

    #region "Search Attributes"
    private void searchAttribsBtn_Click(object sender, EventArgs e)
    {
        string searchString = searchAttribsTxt.Text.ToString();
        FindAllOfMyString(searchString);
    }

    private void FindAllOfMyString(string searchString)
    {
        if (searchString != string.Empty)
        {
            for (int i = 0; i < columnNames.Items.Count; i++)
            {
                if (columnNames.Items[i].ToString().IndexOf(searchString.Trim(), StringComparison.OrdinalIgnoreCase) >= 0)
                {
```

```
            columnNames.SetSelected(i, true);
            columnNames.SetItemChecked(i, true);
        }
        else
            columnNames.SetSelected(i, true);
    }
  }
}
#endregion

#region "Show Attributes"
private void listShowAttributes_DoubleClick(object sender, EventArgs e)
{
  if (listShowAttributes.SelectedIndex > -1)
  {
    string selectedValue = listShowAttributes.SelectedItem.ToString();
    string selectedTable = (selectedValue.IndexOf(".") > -1 ? ((selectedValue.Split(".".ToCharArray()))[0]) : "");
    string selectedColumn = (selectedValue.IndexOf(".") > -1 ? ((selectedValue.Split(".".ToCharArray()))[1]) : selectedValue);
    DataTable dt;
    String Type = string.Empty;

    if (selectedTable.Length > 0)
```

```
        {
            dt = ds.Tables[selectedTable];
            if (selectedColumn.Length > 0)
            {
                conditionsEntry.Text += conditionsEntry.Text.Trim().Length > 0 ? "" : "";
                conditionsEntry.Text += " " + selectedValue + " ";
            }
        }
        listShowAttributes.SelectedIndex = -1;
    }
}
#endregion

#region "Click Operators"
private void operators_DoubleClick(object sender, EventArgs e)
{
    if (operators.SelectedIndex > -1)
    {
        conditionsEntry.Text += conditionsEntry.Text.Trim().Length > 0 ? "" : "";
        conditionsEntry.Text += " " + operators.SelectedItem.ToString() + " ";
    }
    operators.SelectedIndex = -1;
```

```
}
#endregion

#region "Verify Conditions"
private void verifyConditions_Click(object sender, EventArgs e)
{
   bool validation = Validate();
   if (!validation) { MessageBox.Show("No error found. A Valid Condition!"); }
}

public bool Validate()
{
   bool OperatorInd = false;
   bool ValueInd = false;
   bool ColumnInd = false;
   string InvalidCollumns = string.Empty;
   bool errorFound = false;

   //Getting the Where House Conditional String in to a variable.
   string WhereHouseCondition = conditionsEntry.Text.Trim();
   WHC.Clear();
   //Splitting the string in to multiple strings based on 'AND' & 'OR' conditions to get the individual Conditions.
```

```
string[] Conditions = WhereHouseCondition.Split(new string[] { " AND", " OR", " and", " or" },
StringSplitOptions.RemoveEmptyEntries);

for (int CondCnt = 0; CondCnt < Conditions.Count(); CondCnt++)
{
  string[] WHAttr = Conditions[CondCnt].Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

  if (WHAttr.Count() == 2 || WHAttr.Count() == 1)
  {
    for (int WHAttrCnt = 0; WHAttrCnt < WHAttr.Count(); WHAttrCnt++)
    {
      for (int cn = 0; cn < operators.Items.Count; cn++)
      {
        if (!WHAttr[WHAttrCnt].Equals(operators.Items[cn].ToString()))
        {
          if (WHAttr[WHAttrCnt].Contains(operators.Items[cn].ToString()))
          {
            MessageBox.Show("No Space found at " + WHAttr[WHAttrCnt]);
            OperatorInd = true;
            errorFound = true;
          }
        }
```

```
            else if (WHAttr[WHAttrCnt].Equals(operators.Items[cn].ToString()))
            {
               OperatorInd = true;
               ValueInd = true;
            }
         }
      }
      if (OperatorInd && ValueInd) { MessageBox.Show("No value found at " + Conditions[CondCnt]); errorFound = true; }
      if (!OperatorInd) { MessageBox.Show("No Operator Found at " + Conditions[CondCnt]); errorFound = true; }
   }
   if (WHAttr.Count() == 3)
   {
      int Num;
      if (!int.TryParse(WHAttr[2], out Num))
      {
         if (WHAttr[2].Substring(0, 1) == "\"" && WHAttr[2].Substring(WHAttr[2].Length - 1, 1) == "\"") { }
         else if (WHAttr[2].Substring(0, 1) == "'" && WHAttr[2].Substring(WHAttr[2].Length - 1, 1) == "'")
         {
            MessageBox.Show("“All string values must be entered in double quotes.”" + " : " + WHAttr[2]);
            errorFound = true;
         }
         else
```

```
    {
      if (!columnNames.Items.Contains(WHAttr[2]))
      {
        if (WHAttr[2].Contains("."))
          MessageBox.Show("Inavalid Column: " + WHAttr[2]);
        else
          MessageBox.Show(""“All string values must be entered in quotes.”" + " : " + WHAttr[2]);
        errorFound = true;
      }
      else
        ExtraWHTabs.Add(WHAttr[2].Split('.').ToArray()[0].ToString().Trim());
    }
  }
  if (!columnNames.Items.Contains(WHAttr[0]))
  {
    InvalidCollumns = InvalidCollumns + WHAttr[0] + " ";
    ColumnInd = true;
  }
  if (ColumnInd) { MessageBox.Show("Invalid Column Name(s) "); errorFound = true; }// + InvalidCollumns);
  if (!errorFound)
  {
    WHC.Add(new WHConditions(WHAttr[0], WHAttr[1], WHAttr[2]));
```

```
            }
        }
        if (WHAttr.Count() > 3)
        {
            MessageBox.Show("Incorrect Syntax: " + Conditions[CondCnt]); errorFound = true;
        }
    }
    ExtraWHTabs = ExtraWHTabs.Distinct().ToList();
    return errorFound;
}
#endregion

#region "Fix Attributes"
private void btnFixAttr_Click(object sender, EventArgs e)
{
    if (listShowAttributes.Items.Count == 0)
        MessageBox.Show("No attributes selected. Select the attributes first");
    else
    {
        lstFixedAttributes.Items.Clear();
        for (int cnt = 0; cnt < listShowAttributes.Items.Count; cnt++)
        {
```

```
            lstFixedAttributes.Items.Add(listShowAttributes.Items[cnt]);
        }
        this.ClearAttributes_Click(sender, e);
        btnFixAttribs.Enabled = false;
        lblMessage.Text = "Next Step: Select attributes and desired operators from the list to build warehouse condition";
      }
    }
    #endregion

    #region "Reset Fix Attributes"
    private void btnResetFixAttribs_Click(object sender, EventArgs e)
    {
      lstFixedAttributes.Items.Clear();
      btnFixAttribs.Enabled = true;
      lblMessage.Text = "Next Step: Select the required attributes from the list and click on 'Fix Attributes' button";
    }
    #endregion

  #endregion
  }
}
```

Utils.cs – Create for DB connectivity (Future work)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;


namespace QueryInterface
{
  public static class Utils
   {
     public static string getConnectionString(string value)
     {
        string connectionString = "Data Source=(local);Initial Catalog=" + value + ";Persist Security Info=True;User
ID=<dbuser>;Password=<dbpassword>";
        return connectionString;
     }
   }
}
```